Solving Recurrence Relations

CS130A Coakley

Analyze

Some Functions are Easy

```
for(int i = 0; i < n; i++) {
    for(int j = 0; j < n; j++) {
        do_something_simple();
    }
}
```

$O(n)$

$O(n^2)$

No early termination? Done!

Sometimes We Could Be More Precise

```
for(int i = 0; i < n; i++) {
    for(int j = 0; j <= i; j++) {
        do_something_simple();
    }
}
```

$n(n+1)/2$, still $O(n^2)$

Drop one “=”, becomes $n(n-1)/2$, still $O(n^2)$

Recursive Functions Can Be Tricky

```
int pow(int base, int ex) {
// only works on powers of 2
    if (ex == 0) return 1;                               Constant time
    if (ex == 1) return base;                            Constant time
    return pow(base, ex/2) * pow(base, ex/2);            Function of ex
}
```

Recursive Functions Can Be Tricky

```
int pow(int base, int ex) {
// only works on powers of 2
	if (ex == 0) return 1;
	if (ex == 1) return base;
	return pow(base, ex/2) * pow(base, ex/2);
}
```

$T(0) = C$

$T(1) = C$

$T(N) = 2*T(N/2) + C_2$

Assumption that return/mult is constant

Substitute, look for pattern

$T(N) = 2*T(N/2) + C_2$

$T(N) = 4*T(N/4) + 2*C_2$

$T(N) = 8*T(N/8) + 4*C$

$T(N) = 2^k*T(N/2^k) + 2^{k-1}*C_2$

Choose k such that $N/2^k = 1$

$2^k = N, k = \lg N$

$T(N) = 2^{\lg N}*T(1) + 2^{\lg N - 1}C_2$

$T(N/2) = 2*T(N/4) + C_2$

$T(N/4) = 2*T(N/8) + C_2$

$T(N/8) = 2*T(N/16) + C_2$

...

$2^{\lg N - 1} = 0.5*N$

Substitute and Simplify

$T(N) = 2^{\lg N} * T(1) + 2^{\lg N - 1} C_2$

$T(N) = N * C + 0.5 * N * C_2$

$T(N) = O(N)$

Master Theorem

Easy Memorization Way

T(n) = a*T(n/b) + f(n)

a is the number of sub-problems

b is the chunk size (assume equal size)

f(n) is the additional processing

You can memorize solutions for 3 special cases.

Aside, this is covered nicely:
https://en.wikipedia.org/wiki/Master_theorem

Following examples from Wikipedia

Case 1 - Divide and conquer dominates

If $f(n) = O(n^c)$ where $c < \log_b$ **a**

$T(n) = \Theta(n^{\log_b a})$

$T(n) = 8T(n/2) + 1000n^2$

$a = 8$

$b = 2$

$f(n) = 1000n^2$

$\lg 8 = 3 > c$

$T(n) =$ **$\Theta(n^3)$**

Case 2 - It all matters

If, for some $k>=0$, $f(n) = \Theta(n^c lg^k n)$ **where** $c = log_b a$

$T(n) = \Theta(n^c lg^{k+1} n)$

$T(n) = 2T(n/2) + 10n$

$a=2$

$b=2$

$f(n) = 10n$

$c=1$ $(k=0)$

$T(n) = \Theta(n \lg n)$

Case 3 - Your other stuff dominates divide-and-conquer

If $f(n) = \Omega(n^c)$ where $c > \log_b a$

$T(n) = \Theta(f(n))$

$T(n) = 2T(n/2) + n^2$

$a = 2$

$b = 2$

$f(n) = \Omega(n^2)$, $c = 2$, $\log_2 2 = 1$, $2 > 1$

$T(n) = \Theta(n^2)$